Foley Prep Developer Interview: Task 2

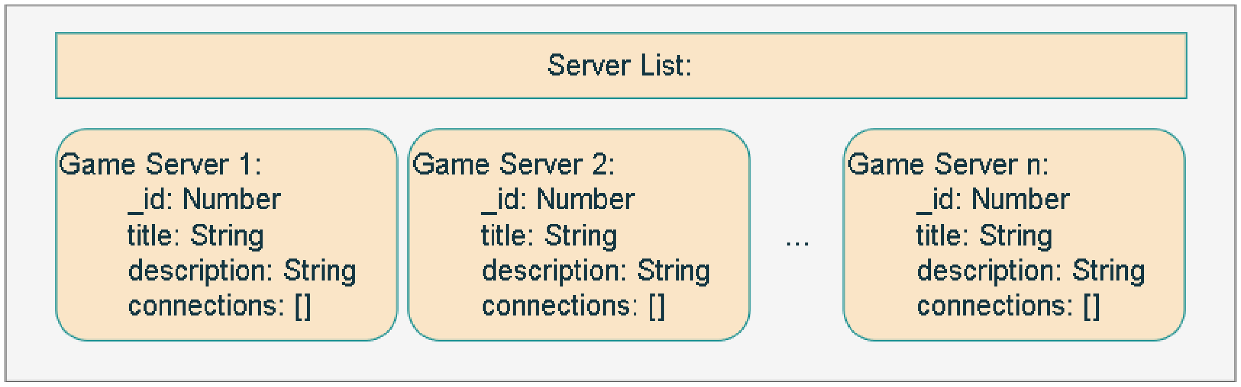
Overview

At a high level, this application needs to facilitate connections between an arbitrary number of both users and lobbies. The application will maintain a list of all open game lobbies. Each entry in that list will contain a list of all connections to that game lobby. When a client connects to a server via websockets, that game server will store the websocket connection in the connection list associated with that lobby. The data structures section below provides more details on these structures.

Data Structures

The main structure required for this implementation is a master list of open servers. Each server in this list would contain an id and a list of all current user connections. Since the number of servers and clients is likely to vary, a list is a better choice than a fixed length array.

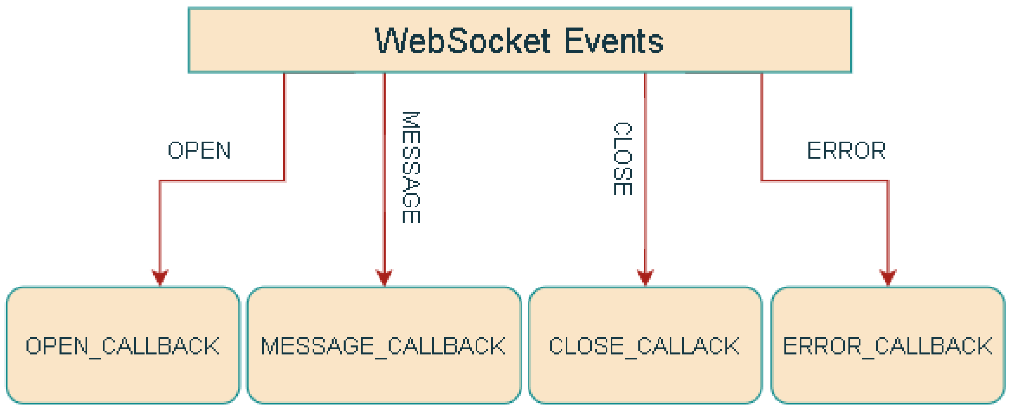
The figure below shows the outline of the server list, including the data stored by each game server.


Server List:
Game Server 1: _id: Number title: String description: String connections: []
Game Server 2: _id: Number title: String description: String connections: []
...
Game Server n: _id: Number title: String description: String connections: []

The fields of each game server are as follows:

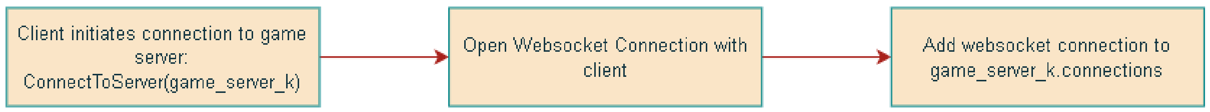
1. _id: A unique id assigned to each server
2. title: The title of the server as it appears to a user
3. description: Any description information for the server
4. connections: The list of websocket connections to that server

Each element in the connections list will be a websocket object. This object maintains four events: OPEN, MESSAGE, CLOSE, and ERROR. Javascript natively provides functionality to map callback functions to each event. These callback functions allow real time maintenance of the master server list mentioned above. For example, when a client tries to open a websocket connection, the open callback function would be triggered.

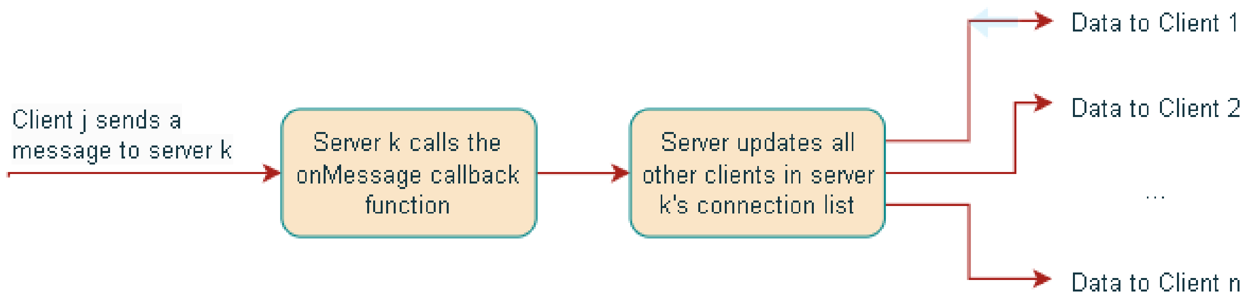


Operation

When a user loads the game they would see an appropriately formatted version of the list of servers. When they select a lobby, their webpage will initiate a websocket connection with the appropriate server using its id. The diagram below shows the process for a user to initiate a websocket connection to a specific server.


Client initiates connection to game server: ConnectToServer(game_server_k)
Open Websocket Connection with client
Add websocket connection to game_server_k.connections

The primary purpose of the websocket is to facilitate continuous two communication. For this purpose, the websocket API provides a send method to transmit data. Using this method, the server can propagate any relevant updates from one client to the others. For example, if game server k receives a message from client j with a new data, server k will transmit this update to all other clients in its connections list (excluding client j). The diagram below demonstrates the server's response to a client response to a client message.


Client j sends a message to server k
Server k calls the onMessage callback function
Server updates all other clients in server k's connection list
Data to Client 1
Data to Client 2
...
Data to Client n